



**VA FILEMAN
SQL INTERFACE (SQLI)
VENDOR GUIDE
*DRAFT***

Patch DI*21.0*38

Fall, 1997

Department of Veterans Affairs
VISTA Software Development
OpenVISTA Product Line

Table of Contents

Orientation.....	iii
Introduction.....	v
1. Building an SQLI Mapper.....	1-1
Information Provided by SQLI	1-2
Organization of SQLI Information.....	1-2
SQLI Entity - Relationship Diagram	1-3
Guidelines for SQLI Mappers.....	1-4
VA Programming Standards and Conventions	1-4
Populating the SQLI_KEY_WORD File	1-4
Data Dictionary Synchronization	1-4
Kernel Compatibility	1-5
2. Parsing the SQLI Projection.....	2-1
About the Examples in this Chapter	2-1
Using the {B}, {E}, {I}, {K}, and {V} Placeholders	2-1
Example File	2-3
Starting Point: SQLI_SCHEMA File	2-4
To Find the Projected Table for a File	2-4
Processing Tables.....	2-4
About Table Elements.....	2-5
Processing Columns	2-6
To Find a Table Element's Column Entry	2-6
Ien Columns	2-7
To Find the Primary Key for a Given Table	2-7
Primary Key for a Projected Subfile	2-8
\$ORDERING to Loop Through a File's Data Entries	2-9
Assembling Record Locations	2-10
Retrieving Column Values.....	2-11
Column Value Conversions	2-12
Domain Conversions (Base to Internal)	2-12
Output Format Conversions (Base to External)	2-12
Foreign Keys	2-13
3. VA FileMan and SQL.....	3-1

Mapping VA FileMan Fields to SQL Data Types.....	3-3
VA FileMan Indexes	3-7
4. File Reference	4-1
SQLI_SCHEMA File	4-2
SQLI_KEY_WORD File	4-3
SQLI_DATA_TYPE File	4-4
SQLI_DOMAIN File	4-5
SQLI_KEY_FORMAT File.....	4-7
SQLI_OUTPUT_FORMAT File.....	4-8
SQLI_TABLE File.....	4-9
SQLI_TABLE_ELEMENT File	4-10
SQLI_COLUMN File	4-11
SQLI_PRIMARY_KEY File	4-14
SQLI_FOREIGN_KEY File	4-16
SQLI_ERROR_TEXT File.....	4-17
SQLI_ERROR_LOG File.....	4-18
5. Entry Points/Supported References.....	5-1
6. Other Issues	6-1
Domain Cardinality	6-1
SQLI and Schemas.....	6-1
SQL Identifier Naming Algorithms	6-2
VA Business Rules and Insert/Update/Delete Operations.....	6-3
SQLI Implementation Notes	6-3
Appendix A: Quick Reference Card.....	A-1
Glossary	Glossary-1
Index	Index-1

Orientation

Typographic Conventions

At some places in this manual, you are shown a simulation of your interaction with your computer. In order to distinguish computer-supplied prompts from your responses, responses are in bold type. Like this:

COMPUTER'S PROMPT: **USER'S RESPONSE**

VA FileMan Information

Additional information about VA FileMan is available on the VA FileMan home page (on the VA intranet):

<http://www.vista.med.va.gov/softserv/infrastr.uct/fileman/>

For information about VA FileMan, consult its documentation set. VA FileMan manuals of particular interest to M-to-SQL vendors are:

VA FileMan V. 21.0 Programmer Manual

VA FileMan V. 21.0 User Manual

These manuals contain detailed information on VA FileMan, including its data dictionary structures, data format, field types, and API calls. They are available in both hardcopy and Adobe Acrobat PDF formats. Manuals in PDF format are available from the VA FileMan home page.

Additional SQLI Information

Additional information about SQLI is available on the SQLI home page:

<http://www.vista.med.va.gov/softserv/infrastr.uct/sqli/>

On that home page, an additional SQLI manual is available, targeted for sites implementing SQLI:

- *VA FileMan SQLI Site Manual*

Introduction

What is VA FileMan?

VA FileMan is a database management system (DBMS) which is used at DVA medical facilities. It is implemented in the M programming language.

With the release of VA FileMan Version 21 in December of 1994, VA FileMan introduced a silent Database Server (DBS) programming API, which set the stage for extending database access to non-host users on local and wide area networks. SQLI, for example, makes extensive use of VA FileMan's DBS API.

What is SQLI?

VA FileMan's SQLI (SQL Interface) product projects a relational view of VA FileMan data dictionaries for use by M-to-SQL vendors. This provides a supported mechanism for M-to-SQL vendors to access VA FileMan's internal data dictionaries. M-to-SQL vendors can use SQLI to map their SQL data dictionaries directly to VA FileMan data. By doing this they view and access VA FileMan data as native SQL tables.

What is the Purpose of this Manual?

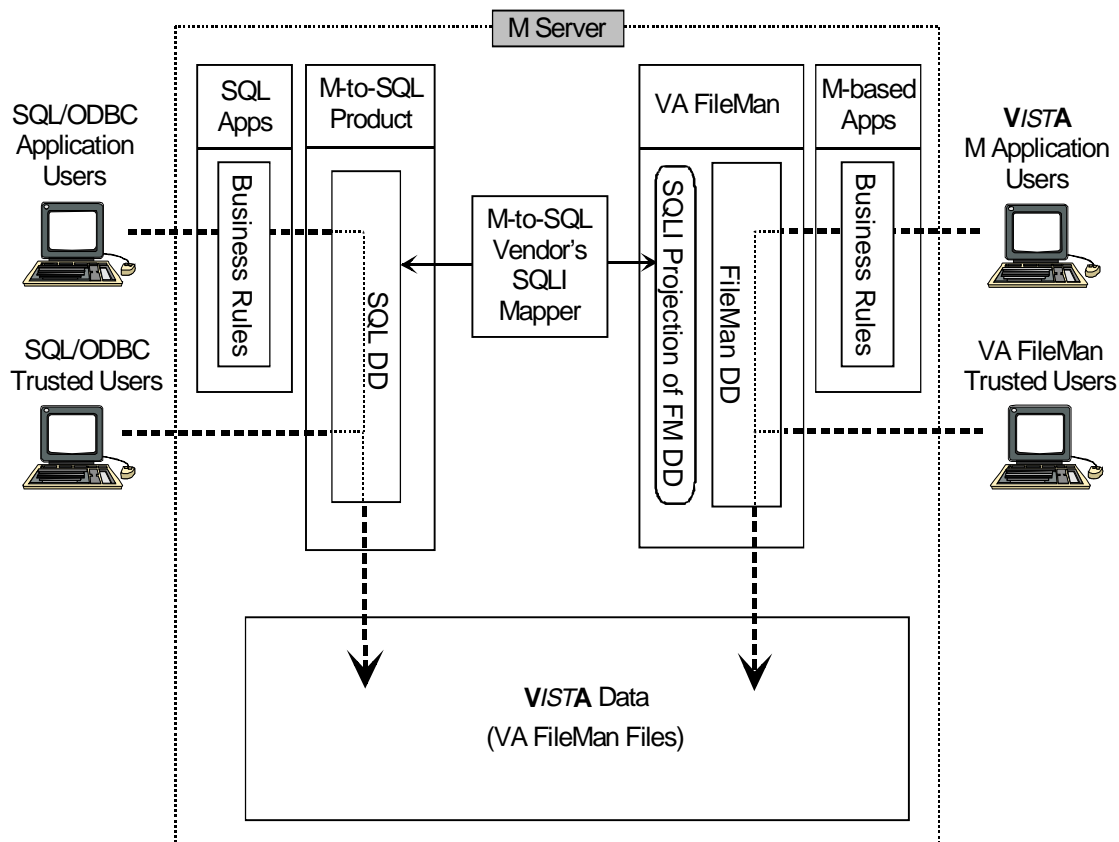
This manual is designed to help you, the M-to-SQL vendor, create and maintain an SQLI mapper utility. An SQLI mapper utility reads the projection of VA FileMan's data dictionaries provided by SQLI. It maps your M-to-SQL product's data dictionaries based on SQLI's projection so that your M-to-SQL product can directly access VA FileMan data as relational tables.

This manual may also be useful if you are providing technical support for an SQLI system; it can help provide an understanding of how SQLI works.

Introduction

1. Building an SQLI Mapper

To map your M-to-SQL product's data dictionaries to directly access VA FileMan data, based on the information projected by SQLI, you will need to create an SQLI mapper utility. This SQLI mapper utility should read the published information on each VA FileMan file from the SQLI's projection. It should use this information to generate DDL commands (or use some similar method) that map your SQL data dictionaries directly to VA FileMan data.



Information Provided by SQLI

SQLI's projection of VA FileMan data dictionaries provides:

- A complete projection of VA FileMan files and fields as relational tables.
- Pre-defined SQL-compatible names for tables, columns, and keys.
- Global locations to retrieve data elements directly.
- Code to retrieve data elements through API calls.
- Code to convert retrieved data elements from internal FileMan format to base and external column formats.
- A standard set of strategies for VA FileMan field types whose projection in relational terms is non-trivial (pointer fields, variable pointer fields, word processing fields, and subfiles).

This information is published in a way that is tailored to use by an M-to-SQL vendor. It relieves you from having to access VA FileMan's internal data dictionary structures to determine certain parameters that are not explicit in VA FileMan. Also, using SQLI should insulate your code from proposed changes in the VA FileMan data dictionary.

Organization of SQLI Information

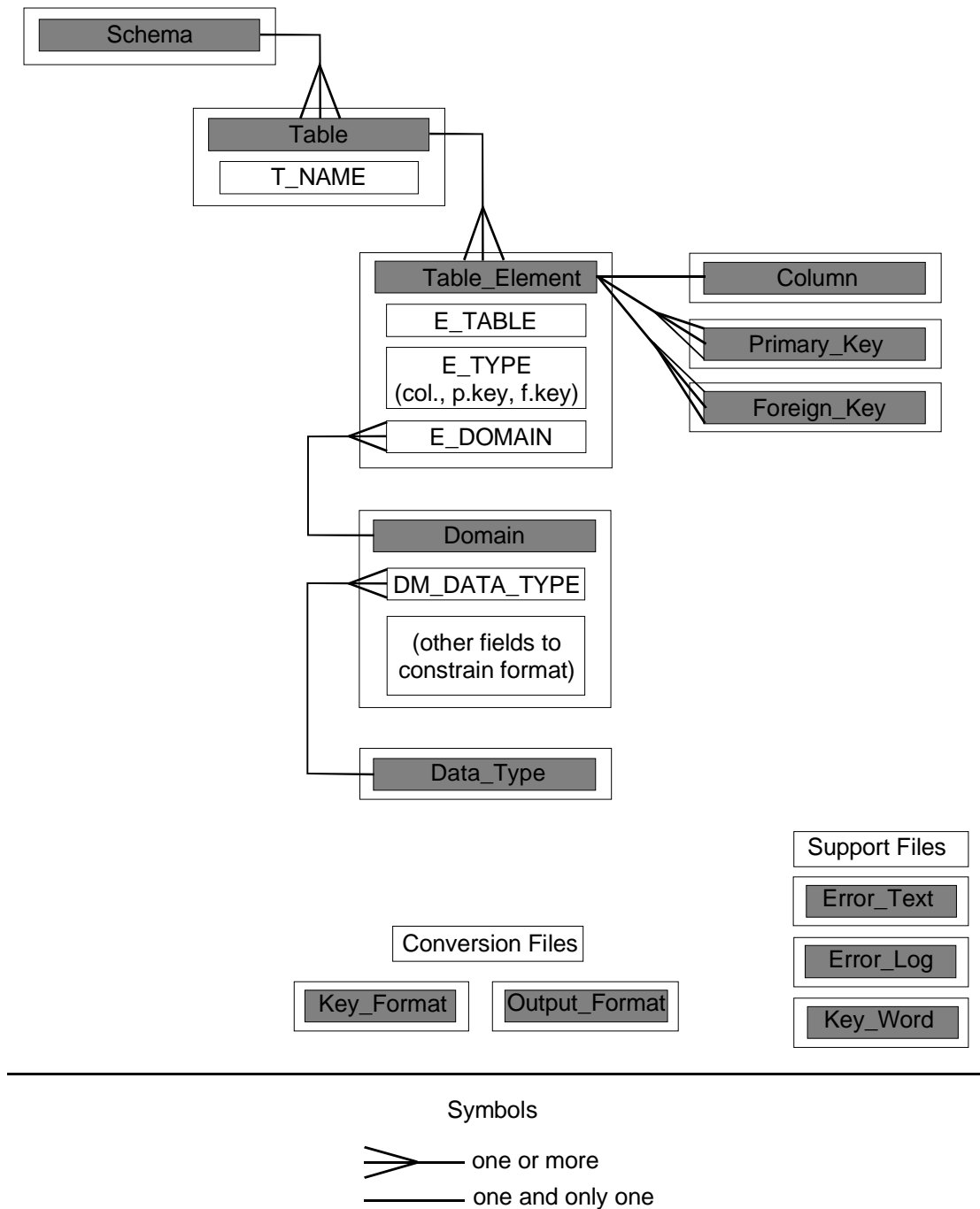
SQLI is implemented as a set of VA FileMan files within a single M global, with no multiples or word processing fields.

The organization of the files mirrors SQL2 standard Data Definition Language (DDL) syntax. Every data structure in the main SQLI files reflects some portion of the DDL commands needed to create SQL data dictionaries for VA FileMan data (essentially, the CREATE TABLE command).

Additional syntax has been added to support the definition of M global structures, virtual columns, key and output formats and other objects outside the scope of the SQL standard.

SQLI Entity - Relationship Diagram

This diagram organizes the file entities in their importance to the operation of the SQLI package. It shows conceptual relationships between the files, but not a comprehensive view of the physical pointer relationships between files.



Guidelines for SQLI Mappers

VA Programming Standards and Conventions

Be aware that your code will be running in VA production accounts along with VA code. Adherence to the VA Programming SAC (Standards and Conventions) is highly recommended. This includes guidelines about the setting and killing of variables, the ways that devices are used, and not interfering with the error trapping provided by VA's Kernel package.

Obtaining a formal namespace from the VA's DBA (Database Administrator) is also advised.

Populating the SQLI_KEY_WORD File

The SQLI_KEY_WORD file stores any words that SQLI should not use for SQL entity names. At any given site, it may not be populated with any keywords at all. So you (the M-to-SQL vendor) should use SQLI's KW^DMSQD entry point to populate this SQLI_KEY_WORD file with:

- Any keywords specific to your (vendor) M-to-SQL product
- The standard set of reserved keywords for SQL as defined by the ANSI standard for SQL
- The keywords for ODBC as defined by Microsoft

Also, in your instructions to sites using your SQLI mapper, make sure that adding your keywords to the SQLI_KEY_WORD file is done **prior** to the site generating their first SQLI projection.

Data Dictionary Synchronization

To aid sites with data dictionary synchronization, your SQLI mapper utility should provide entry points for the following functions:

- Remapping your SQL data dictionary for **all** tables projected by SQLI.
- Remapping your SQL data dictionary for **one** table projected by SQLI.

Kernel Compatibility

Besides conforming to the VA Programming SAC, be aware that sites will probably want to run your utilities as background tasks using Task Manager, a module of VA's Kernel package. Sites are likely to want to create a single "task" that calls your keyword utility, runs the VA SQLI projection, and then runs your SQLI mapper.

To be compatible with running as a background task in TaskMan, your keyword utility and SQLI mapper should:

- Not issue any READs or in any way make either entry point interactive. This allows the entry point to run in the background. If you need to ask questions, separate that section of code from the actual SQLI mapper code.
- Not issue USE commands. The "current device" is already opened and available when an entry point is run as a task in the Kernel environment. If you need to use USE commands (for example, to write to a host file), make sure you store the value of the current device so you can return to it.
- For output, issue WRITE commands. Don't use escape sequences, however; any output should be able to print on a simple line printer.

See the Task Manager section of the *Kernel Systems Manual* for more information on background tasks in the Kernel environment.

2. Parsing the SQLI Projection

This chapter gives examples of how to traverse SQLI's indexes and retrieve the information needed to map your SQL data dictionaries.

Retrieving the information stored in the SQLI files involves traversing their indexes and retrieving the field values stored in their indexes and in the entries themselves. Full descriptions of the SQLI file and index structures are contained in the "File Reference" chapter. You may also want to refer to the Quick Reference Card provided in Appendix A.

The global location of each SQLI file and its associated fields and indexes are stable, supported references. You can reference these locations directly.

About the Examples in this Chapter

The specific approaches provided in this chapter are suggestions only, and do not cover all of the ways you can retrieve information from SQLI.

Using the {B}, {E}, {I}, {K}, and {V} Placeholders

SQLI provides M executable code and expressions in certain fields. This M code provided by SQLI can use the following placeholder symbols:

Symbol	Usage
{B}	Base value of a column - used for computation
{E}	External value of a column - used for display
{I}	Internal value of a VA FileMan field - used for storage
{K[1..n]}	Key value - {K} is the current key, {K1} is the first key, etc.
{V[1..n]}	Value - used for function arguments and output value

Field Value Placeholders: {I}, {B} and {E}

- The {I} placeholder is used to represent Internal values, that is, the VA FileMan internal value of a field.
- The {B} placeholder is used to represent Base values, that is, the base value of a column.
- The {E} placeholders is used to represent External values, that is, the externally formatted view of the field that a user should see.

Key Placeholders: {K1}, {K2}, etc.

These placeholders represent portions of the primary key for a table column, numbered corresponding to the P_SEQUENCE values of a primary key. They are used primarily in the C_FM_EXEC field of the SQLI_COLUMN file. Substitute the appropriate primary key values to assemble a global reference to retrieve a particular column value.

For example:

```
^DMSQ("C",672,3) = S {V}=$$GET^DMSQU(9.4901,"{K3},{K2},{K1}",.03)
```

In this case, {K3} represents the value of the part of the primary key whose P_SEQUENCE is 3; {K2} represents the part of the primary key whose P_SEQUENCE is 2; and {K1} represents the part of the primary key whose P_SEQUENCE is 1. This call retrieves the value of a column from its corresponding VA FileMan field.

Return Value Placeholder: {V}

This placeholder is used to denote where to place a variable that should receive a return value. One example of where the {V} "value" placeholder is used is in the SQLI_COLUMN file, in M code provided by the C_FM_EXEC field. For example:

```
^DMSQ("C",485,3) = S {V}=$$GET^DMSQU(1.1,"{K1}",.04)
```

In this case, substitute the variable name you want the output of the \$\$GET function returned in, for the {V} placeholder, before executing the M code.

Example File

Throughout the chapter, a simple VA FileMan file, DA RETURN CODES, is projected by SQLI. Here is a condensed VA FileMan data dictionary listing of this file:

```
CONDENSED DATA DICTIONARY---DA RETURN CODES FILE (#3.22)
UCI: VAH,FLD   VERSION: 8.0
STORED IN: ^%ZIS(3.22,
```

```
-----
FIELD      FIELD
NUMBER     NAME

.01        DA Return String (RF), [0;1]
2          Terminal Type String (RFX), [0;2]
3          DESCRIPTION (Multiple-3.223), [1;0]
          .01 DESCRIPTION (WL), [0;1]
```

Here is a global map VA FileMan data dictionary listing of this file:

```
GLOBAL MAP DATA DICTIONARY #3.22 -- DA RETURN CODES FILE
STORED IN ^%ZIS(3.22, (15 ENTRIES)   SITE: KERNEL   UCI: KRN,KDE
```

```
-----
This file holds the translation between the ANSI DA return code and the
name in the terminal type file that should be used.
```

```
CROSS REFERENCED BY: DA Return String(B), DA Return String(B1)
```

```
^%ZIS(3.22,D0,0)= (#.01) DA Return String [1F] ^ (#2) Terminal Type String
                ==>[2F] ^
^%ZIS(3.22,D0,1,0)=^3.223^^  (#3) DESCRIPTION
^%ZIS(3.22,D0,1,D1,0)= (#.01) DESCRIPTION [1W] ^
```


Starting Point: SQLI_SCHEMA File

This version of SQLI maps all VA FileMan files to a single schema, SQLI. So for the time being, you can assume that all tables are projected within the same schema (SQLI). Therefore, your starting point when processing the information in SQLI should be the SQLI_TABLE file (not the SQLI_SCHEMA file).

In the future, however, SQLI may project tables in more than one schema. At that point in time, an index may be added on the T_SCHEMA field of the SQLI_TABLE file, such that you can loop through schemas, and within schemas process tables.

To Find the Projected Table for a File

Within a given schema, you CAN loop through each table and process the information for that table.

To find the SQLI_TABLE entry for a particular VA FileMan file, you can look up the file's number in the "C" cross-reference of the SQLI_TABLE file. For example, to determine the corresponding SQLI_TABLE entry for the DA RETURN CODES file (#3.22), do:

```
> W $O(^DMSQ("T","C",3.22,""))
97
```

Therefore the internal entry number (ien) of the SQLI_TABLE entry for DA RETURN CODES is 97. That entry in the SQLI_TABLE file looks like:

```
NUMBER: 97                                T_NAME: DA_RETURN_CODES
T_SCHEMA: SQLI
T_COMMENT: This file holds the translation between the ANSI DA return c
T_VERSION_FM: 1                            T_FILE: DA RETURN CODES
T_UPDATE: JUL 31, 1997                     T_GLOBAL: ^%ZIS(3.22,{K})
```

Processing Tables

When processing a table, once you have the table's ien in the SQLI_TABLE file, the next thing to do is loop through the set of table elements for that table.

One way to find the table elements for a given SQLI_TABLE entry is to look up that entry's ien in the "D" index of the SQLI_TABLE_ELEMENT file, and find each matching table element:

```
S EL="" F S EL=$O(^DMSQ("E","D",tableien,EL)) Q:EL']""
```


However, using the "F" index of the SQLI_TABLE_ELEMENT file, you can see both how many and also what type of table elements were projected for a table.

For example, in the case of the DA_RETURN_CODES table (ien #97):

```
Global ^DMSQ("E","F",97
      DMSQ("E","F",97
^DMSQ("E","F",97,"C",256) =
^DMSQ("E","F",97,"C",2273) =
^DMSQ("E","F",97,"C",2274) =
^DMSQ("E","F",97,"C",2275) =
^DMSQ("E","F",97,"P",255) =
Global ^
```

This shows that five table elements (four columns and one primary key) are projected for the DA_RETURN_CODES table.

About Table Elements

Every entry in the SQLI_TABLE_ELEMENT file is associated with at least one entry in the SQLI_COLUMN, SQLI_PRIMARY_KEY, or SQLI_FOREIGN key file. The associated entries contain the details of each table element, and associate themselves with table elements by pointing to the SQLI_TABLE_ELEMENT file.

For columns, only a single column in the SQLI_COLUMN file will point to any given column-type table element.

For primary keys however, one or more entries in the SQLI_PRIMARY_KEY file will point to the single primary key table element for any given table. This is because some primary keys have many parts. Pointing to a single primary key table element is how these many parts in the SQLI_PRIMARY_KEY file are organized into a single comprehensive primary key.

Likewise for foreign keys, one or more entries in the SQLI_FOREIGN_KEY file will point to the single foreign key table element for any given foreign key.

Processing Columns

Let's look at the column-type table element entries for the DA_RETURN_CODES table. These provide the relational specifications for each table element:

NUMBER: 256	E_NAME: DA_RETURN_CODES_ID
E_DOMAIN: INTEGER	E_TABLE: DA_RETURN_CODES
E_TYPE: Column	
E_COMMENT: Primary key #1 of table DA_RETURN_CODES	
NUMBER: 2273	E_NAME: DA_RETURN_STRING
E_DOMAIN: CHARACTER	E_TABLE: DA_RETURN_CODES
E_TYPE: Column	
E_COMMENT: This field holds the string returned from sending a ANSI DA to	
NUMBER: 2274	E_NAME: TERMINAL_TYPE_STRING
E_DOMAIN: CHARACTER	E_TABLE: DA_RETURN_CODES
E_TYPE: Column	
E_COMMENT: This is the string that should be used in a lookup to the terminal type	
NUMBER: 2275	E_NAME: DESCRIPTION
E_DOMAIN: WORD_PROCESSING	E_TABLE: DA_RETURN_CODES
E_TYPE: Column	
E_COMMENT: The description of the description field is that of holding the description	

To Find a Table Element's Column Entry

For table elements that correspond to columns, use the "B" index of the SQLI_COLUMN file to find the corresponding column entry in SQLI.

For example, for the column-type table element entry #2273, the corresponding column is:

```
> W $O(^DMSQ("C","B",2273,""))
1734
```

This entry, in the SQLI_COLUMN file, looks like:

NUMBER: 1734	C_TABLE_ELEMENT: DA_RETURN_STRING
C_WIDTH: 70	C_FILE: 3.22
C_FIELD: .01	C_NOT_NULL: Required
C_SECURE: Not secure	C_VIRTUAL: Base column
C_PARENT: DA_RETURN_CODES_ID	C_PIECE: 1
C_GLOBAL: ,0)	

Ien Columns

SQLI projects one internal entry number (ien) column for every top-level VA FileMan table. This column is intended to be used by you to store the ien of each record. This ien is important for a number reasons, one of which is that SQLI projects the primary key of each table based on the ien column. So you need provide ien columns for each table. In the case of the DA_RETURN_CODES table, the ien column is the DA_RETURN_CODES_ID column.

For subfiles, one ien column is projected in SQLI for each of the subfile's parents. This allows the projected table to store the ien for each "parent" file entry as these entries exist in VA FileMan. This allows end-users to reassemble the relationships in SQL for a subfile table that exist in VA FileMan.

To Find the Primary Key for a Given Table

Use the "F" index in the SQLI_TABLE_ELEMENT file, and search for the single entry with a type of "P":

```
S PKEY=$O(^DMSQ("E","F",tableien,"P",""))
```

This returns a single entry in that represents the primary key of the table in question. In the case of the DA_RETURN_CODES table, the primary key is:

```
> W $O(^DMSQ("E","F",97,"P",""))
255
```

There is only one entry in the SQLI_TABLE_ELEMENT file for a table's primary key. The way a primary key is projected in SQLI is that one or more corresponding entries in the SQLI_PRIMARY_KEY file contain the actual parts of the primary key. They all point back to the single entry in the SQLI_TABLE_ELEMENT file to compose a single, combined primary key. Each SQLI_PRIMARY_KEY entry's P_SEQUENCE field identifies the order in which that part of the primary key should be assembled.

Let's look at the primary key projected for the DA_RETURN_CODES table. Use the SQLI_PRIMARY_KEY file's "B" index to discover how many parts are in the DA_RETURN_CODE file's primary key, based on its primary key table element:

```
Global ^DMSQ("P","B",255
      DMSQ("P","B",255
^DMSQ("P","B",255,159) =
Global ^
```


In this case, the primary key is a single-part key. That entry looks like:

```
NUMBER: 159                                P_TBL_ELEMENT: DA_RETURN_CODES_PK
P_COLUMN: DA_RETURN_CODES_ID                P_SEQUENCE: 1
P_START_AT: 0                               P_END_IF: '{K}'
```

Each part of the primary key, as stored in the SQLI_PRIMARY_KEY file, points to the column upon which that part of the primary key is based. In this case, this part of the primary key (which is the only part) is based on the ien column for the table.

Primary Key for a Projected Subfile

The DA RETURN CODES file contains a word processing field, which is stored like a subfile by VA FileMan. Therefore its primary key has more than one part.

If the ien in the SQLI_TABLE file for the DA_RET_CODES_DESCRIPTION file is 98, then the entry in the SQLI_TABLE_ELEMENT file for its primary key can be obtained as follows:

```
> W $O(^DMSQ("E","F",98,"P",""))
257
```

The matching entries in the SQLI_PRIMARY_KEY file are:

```
Global ^DMSQ("P","B",257
      DMSQ("P","B",257
^DMSQ("P","B",257,160) =
^DMSQ("P","B",257,161) =
```

These entries look like:

```
NUMBER: 160
P_TBL_ELEMENT: DA_RET_CODES_DESCRIPTION_PK
P_COLUMN: DA_RETURN_CODES_ID                P_SEQUENCE: 1
P_START_AT: 0                               P_END_IF: '{K}'

NUMBER: 161
P_TBL_ELEMENT: DA_RET_CODES_DESCRIPTION_PK
P_COLUMN: DA_RET_CODES_DESCRIPTION_ID P_SEQUENCE: 2
P_START_AT: 0                               P_END_IF: '{K}'
```

These are the two parts to the DA_RET_CODES_DESCRIPTION table's primary key.

P_COLUMN for sequence 1 of the primary key points to the ien column in the subfile table that stores the ien of what, in VA FileMan, would be the subfile's parent entry. P_COLUMN for sequence 2 of the primary key points to the ien

column in the subfile table that stores the ien of what, in VA FileMan, would be the ien of the subfile entry.

Therefore, the primary key for the subfile's table combines the ien of entries in each VA FileMan file level above the subfile's table, plus the ien column of the subfile's table itself.

\$ORDERING to Loop Through a File's Data Entries

The P_START_AT and P_ENDIF fields in the SQLI_PRIMARY_KEY file provide the initial value for a \$ORDER loop through a file's actual data entries and the expression to complete the loop.

The example below assumes that the table only contains a single element in the primary key (i.e., the table is for a top-level VA FileMan file). The loop would need to be more complex to loop through entries for a subfile.

```
;IEN      = internal entry number of record to retrieve
;PSTARTAT = P_START_AT value for table's single-part primary key.
;PENDIF   = P_END_IF value for table's single-part primary key.
;DMG      = global storage for entries in this table. It is assumed
;          to be a top-level table, with a single-part primary key.
;
S IEN=PSTARTAT,EXIT=$P(PENDIF,"{K}")_"IEN"_$P(PENDIF,"{K}",2)
F  S IEN=$O(@($P(DMG,"{K}")_IEN_")) D  I @EXIT Q
.I @EXIT Q
.;code to retrieve entry would go here
.W !,IEN
```


Assembling Record Locations

You can assemble the global location of any record given the following pieces of information:

- Each primary key entry in the SQLI_PRIMARY_KEY file for the table
- For each primary key entry, the C_GLOBAL value of the corresponding column
- The column values for each column the primary key is based on

Combine in order of P_SEQUENCE the C_GLOBAL value for each column that is part of a table's primary key. You end up with a string that that is a full global reference, with placeholders for each ien. For example:

```
^DPT( {K} , .373 , {K} )
```

The following sample routine loops through each column in a table's primary key in order of P_SEQUENCE, retrieves the C_GLOBAL value for each column, and assembles the global reference for file entries for that table:

```
;      DMT: table number in question
;      DMK: placeholder string
;      DMEP: primary key element
;      DM: primary key column sequence (P_SEQUENCE)
;      DMC: column for a part of the primary key
;      DMCG: C_GLOBAL value for column
;      DMG: accumulated global root
;
S DMK="{K}" , DMG=""
S DMEP=$O(^DMSQ("E","F",DMT,"P", ""))
S DM=0 F S DM=$O(^DMSQ("P","C",DMEP,DM)) Q:DM="" D
. S DMS=DM,DMC=$O(^DMSQ("P","C",DMEP,DM, ""))
. S DMCG=^DMSQ("C",DMC,1),DMG=DMG_DMCG_DMK
S DMG=DMG_" " W DMG
```

The string you generate will look exactly like the value in the SQLI_TABLE file's T_GLOBAL field.

To determine the storage location of a particular entry in that table, replace the {K} placeholders with the value of each part of the primary key for the entry. In the above example, the first {K} would be replaced by the part of the subfile's primary key whose P_SEQUENCE is 1, and the second {K} with the part of subfile's primary key whose P_SEQUENCE is 2.

Retrieving Column Values

Each VA FileMan field type except computed has a fixed global storage location within each corresponding VA FileMan entry. Appending the value in a column's C_GLOBAL field to the storage location of the record in question yields the node that the corresponding field is stored in.

- For fields using normal storage, SQLI provides the ^-delimited piece of the data node in the C_PIECE field.
- For fields using extract storage, SQLI provides the extract from and extract to positions for the data node in the C_EXTRACT_FROM and C_EXTRACT_THRU fields.

Data you retrieve from VA FileMan data globals is in internal VA FileMan format. Sometimes you can use this data without conversions of any kind. However:

- Domain conversions are provided when the internal VA FileMan format differs from the base column format (see Column Value Conversions below).
- Output formats are provided for columns whose external format differs from the base column format (see Column Value Conversions below).

Retrieving Column Values through a DBS Call

The SQLI_COLUMN file provides code in the C_FM_EXEC field to retrieve the external field value a DBS call, for columns derived from the following VA FileMan field types:

- Computed
- Pointer
- Variable Pointer

This code is useful for resolving the external value for pointer field types. A pointer field in one file can point to a pointer field in another file and so forth, resulting a long pointer chain until you finally reach a non-pointer field to access the external value of the original pointer field.

Also, a DBS call is also the only way to retrieve the value for computed fields, which have no permanent storage. A value of 1 in the C_VIRTUAL field indicates which columns are based on computed fields. For such columns, use the M code in the C_FM_EXEC field to retrieve the computed field value.

Column Value Conversions

SQLI provides column conversions for some columns. Base-to-internal conversions are provided in the SQLI_DOMAIN file. Base-to-external conversions are provided in the SQLI_OUTPUT_FORMAT file.

Domain Conversions (Base to Internal)

Some domains created by SQLI provide conversions between VA FileMan internal {I} format to SQL base {B} data format. No conversion is provided when the SQL base and VA FileMan internal form for a column are the same.

Specifically, for columns whose domains are date-time valued (FM_DATE and FM_MOMENT), the domains in the SQLI_DOMAIN file provide conversions in the DM_BASE_EXEC and DM_INT_EXEC fields. Also, the FM_BOOLEAN domain provides conversions in the DM_INT_EXPR and DM_BASE_EXPR fields.

You should always check the domain file when processing columns to determine if a domain conversion is provided.

Output Format Conversions (Base to External)

Given the base column value derived from a VA FileMan field, entries in the SQLI_OUTPUT_FORMAT file provide M code to generate the external value to present to the end-user for the column in question.

Columns don't need an output format if the **base** column data format is the same as its **external** data format. Output formats are therefore provided only for columns derived from Pointer and Set of Codes VA FileMan field types.

Output formats that affect a column can be designated for individual columns, for all columns in a given SQLI_DOMAIN domain, and for all columns whose domain is a given SQLI_DATA_TYPE data type. The order of precedence for which output format to use, if there is more than one, is as follows:

1. C_OUTPUT_FORMAT in the column's SQLI_COLUMN entry
2. DM_OUTPUT_FORMAT in the associated domain's SQLI_DOMAIN entry
3. D_OUTPUT_FORMAT in the associated data type's SQLI_DATA_TYPE entry

You should always check the SQLI_OUTPUT_FORMAT file when processing columns to determine if an output format conversion is provided.

Foreign Keys

Your M-to-SQL product may or may not support foreign keys. If it does, you can use the foreign keys projected by SQLI to make it easier for the end-user to recreate certain relationships that are explicit in the original VA FileMan data.

SQLI projects foreign keys in the following standard situations:

Situation	Foreign Key(s) Provided
Column based on pointer field	In the table containing the pointer field column, one for the pointed-to file, named <i>pointer_field_name_FK</i> . The join is from the pointer field to the pointed-to table.
Table projected for subfile or word processing field	In the subfile or word processing field's table, one for each parent table, each named <i>parent_table_PFK</i> . Each join links the subfile to its original VA FileMan parent.

One advantage of foreign key syntax over joins is that rows are not lost when the value of a join column is null. For example, foreign key syntax (e.g., `NEW_PERSON_FK@NAME`) can be used in the select clause to obtain the value of the column NAME from the NEW_PERSON table, rather than doing a join to NEW_PERSON in a where clause. A row is returned even if the NAME column of the corresponding row in the NEW_PERSON file is null.

To find all of the foreign keys for a given table, use the "F" index of the `SQLI_TABLE_ELEMENT` file, and search for all entries with a type of "F":

```
S COL="" F S COL=$O(^DMSQ("E","F",tableien,"F",COL)) Q:COL']"
```


Pointer Fields

In the case of foreign keys set up to mimic the relationship provided by **pointer** fields, the name of the foreign key is the pointer field's name followed by "_FK". For example:

Pointer field column: TEMPORARY_STATE
Pointer field from table: NEW_PERSON
Pointer field to table: STATE
Foreign key name: TEMPORARY_STATE_FK

Subfiles and Parent Foreign Keys

Tables derived from **subfiles**, including those for word processing fields, have foreign keys projected by SQLI to each table that is a higher file level (up to the top-level file that is the highest parent of the subfile). These foreign keys within a subfile's table are named with the pointed-to table name followed by "_PFK" (parent foreign key). For example:

Subfile table: NEW_PERSON_ALERT_DATE_TIME
Parent table: NEW_PERSON
Foreign key name: NEW_PERSON_PFK

Every foreign key to a given table has the same domain as the primary key of that table. While not supported by SQL, this convention makes entity relationships more explicit and should help vendors maintain referential integrity constraints during mapping.

3. VA FileMan and SQL

VA FileMan, SQL and the Relational Model

The following table lists the equivalent terminology between VA FileMan (projected as a relational database), SQL, and the relational model.

VA FileMan	SQL	Relational Model
File or Multiple	Table	Relation
Field	Column	Attribute
Label	Name	Name
Field Type	Domain	Domain
Record	Row	Tuple

VA FileMan File Definition Structures

The entities that together form a VA FileMan file definition (data dictionary) are contained at the following locations:

Data Dictionary Element	Location
Dictionary of Files	^DIC(Filenumber,
Attribute Dictionary	^DD(Filenumber,
Field Definition Nodes	^DD(Filenumber, fieldnumber,
File Header	Zero subscript of the file's global root

You should not need to access any of this information directly. All relevant information about file definitions needed for projecting VA FileMan data is published by SQLI. For more information on file definition structures, see the Global File Structure chapter of the *VA FileMan Programmer Manual*.

VA FileMan Field Types

The following table lists each of the 9 possible VA FileMan field types. More information on the specifics of each field type can be found in the *VA FileMan V. 21.0 User Manual*.

Field Type	Description
Computed	Value is computed on-the-fly (no permanent storage)
Date	Time can be mandatory, optional, or not allowed
Free Text	Free Text, up to 250 characters in length
MUMPS	Contains MUMPS code
Numeric	Can be integer or decimal-valued
Pointer	Points to .01 field of an entry in another file (value is ien of pointed-to entry)
Set of codes	Restricts a user to just a few possible values. Codes have an internal and external format.
Variable Pointer	Like a pointer field, except that the pointer may be to an entry in one of several files.
Word Processing	This is a memo-type field, with no size limit, implemented in a subfile-like structure. It stores multiple lines of text, and has no size limit.

VA FileMan Subfiles (Multiples)

VA FileMan entries can contain "multiple-valued" fields, known as multiples or subfiles. A subfile is essentially a file-within-a-file. For example, a patient file entry might have an "Appointments" multiple-valued field. This file-within-a-file can contain one or more entries for the patient's appointments. Multiples can themselves contain multiple-valued fields.

Viewed from within VA FileMan, multiples are hierarchical. Data storage for an entry's multiple field is contained descendant from the same subscript as data for the entry itself. However, it is possible to conceptually "flatten" multiples and project them as if they are standalone tables, especially since they are defined in a similar fashion to standalone files in VA FileMan's attribute dictionary. SQLI handles multiples in this fashion.

Mapping VA FileMan Fields to SQL Data Types

VA FileMan field types don't correspond exactly to the SQL concept of data types, but are projected in ways that ultimately result in categorization by data type.

You can determine the original VA FileMan field type of a column through the associated domain's DM_FILEMAN_FIELD_TYPE field. This is a set of codes field, the value of which represents the original VA FileMan field type of the column (and domain) in question.

Ien Column

SQLI provides a column for the original ien of each VA FileMan record. The name for the ien column is based on the table name followed by "_ID". For example, the PATIENT file has a single column primary key, PATIENT_ID.

Computed Fields

Projection of computed fields is complicated mildly by the fact that SQL DDL syntax supports only base data, while Data Manipulation Language supports expressions. Columns for VA FileMan computed fields are flagged with the C_VIRTUAL field in the SQLI_COLUMN file. You can retrieve their computed value with the code in each column's C_FM_EXEC field, which uses DBS calls.

A number of different computed field return value types are possible: Multiline, Boolean-valued, Free text, Date, and Numeric.

Note that Multiline computed fields are not supported by the DBS or by SQLI; a character error message is returned by the SQLI-provided M code as the value for a multiline computed field.

Date Fields

Code is provided in the two FileMan-specific date domains, FM_DATE and FM_MOMENT, to convert between internal VA FileMan formatted dates and date/times, and column "base format" SHOROLOG dates and date/times. The code is in the DM_INT_EXEC and DM_BASE_EXEC fields.

Free Text, Numeric, and MUMPS Fields

No conversion is needed for these three field types; internal, base, and external formats are identical.

Pointer Fields

The pointer field type conforms to SQL's foreign key constraint, and is projected as such in SQLI. VA FileMan, however, allows direct reference to a pointer field, returning the text value of the primary identifier of the row reached by recursively following the pointer chain until the identifier is not itself a pointer. This usage is projected in SQLI by giving pointers a numeric domain and an output format that uses the DBS to return the resolved value.

For example:

```
OF_NAME: FOREIGN_FORMAT_PTOF          OF_DATA_TYPE: INTEGER
OF_COMMENT: Output format for pointer to FOREIGN_FORMAT
OF_EXT_EXPR: $S(' {B}:" ",1:$GET^DMSQU(.44,{B}_",",.01))
```

Substitute the base value of the column for {B}, and the expression returns the resolved external text value of the pointer field.

Set of Codes Fields

An output format is provided for **each** distinct set of codes "set" to display the long form of the base column value (which should be the code only). These output format entries are pointed to from SQLI_COLUMN file entries.

For example:

```
OF_NAME: M_MERGE_O_OVERWRITE          OF_DATA_TYPE: CHARACTER
OF_COMMENT: Set output format
OF_EXT_EXPR: $P($P(" ;m:MERGE;o:OVERWRITE;" , ";"_ {B}_ ":" , 2) , ";" )
```

Substitute the base value of the field (which the same as its VA FileMan internal form for Set of Codes field types) for {B}, and the expression returns the external value of the code.

Variable Pointer Fields

The variable pointer data type is not relationally atomic, the only true violation of the relational model in VA FileMan. In SQLI, a column for a variable pointer field has a character domain, and an output format that returns the VA FileMan display value from whichever of the VA FileMan files each entry actually points to.

Summary: How SQLI Translates FileMan Field Types into SQL Columns

FM Field Type	FM Internal Format	SQL Domain, Data Type, Base Format	SQL External Format
Computed	Date-valued: (see Date FM Field type).		
	Numeric valued: (see Numeric FM Field type).		
	Multiline-valued	CHARACTER domain/data type.	Null.
	Free Text and Boolean-valued (see Free Text FM Field type).		
Date	yyymmdd.hhmmss yyy: #yrs. since 1700 mm: month (00-12) dd: day (00-31) hh: hour (00-23) mm: minute (01-59) ss: seconds (01-59)	Date only: FM_DATE domain; DATE data type. Date w/Time optional: FM_MOMENT domain, MOMENT data type. Date w/Time required: FM_DATE_TIME domain, MOMENT data type. Base format is date/time in SHOROLOG format.	User-friendly version of date. For example, JUL 31, 1997
Free Text	Free text.	CHARACTER domain, data type. Base format: same as FM internal format.	Same as base format.
MUMPS	Free text.	FM_MUMPS domain, CHARACTER data type. Base format: same as FM internal format.	Same as base format.
Numeric	Numeric.	NUMERIC or INTEGER domain and data type. Base format: same as FM internal format.	Same as base format.
Pointer	Numeric ien of the pointed-to entry.	POINTER domain. NUMERIC data type.	External .01 field value of pointed-to entry (pointer chain must be followed) (provided by an output format).
Set of codes	Internally stored "code", typically shorter than the external form.	SET_OF_CODES domain; CHARACTER data type. Base format: same as FM internal format.	External value that the code stands for (provided by an output format).
Variable Pointer	Ien;global file root For example: 4;DIC(42,	VARIABLE_POINTER domain; CHARACTER data type. Base format: External .01 field value of pointed-to entry at end of pointer chain.	External .01 field value of pointed-to entry (pointer chain must be followed) (provided by an output format).
Word Processing	Memo-type field, no size limit, stored in a subfile.	WORD_PROCESSING domain and data type. Base format: A set of rows in a table, one row per textline.	Optionally make available as a memo field; otherwise, same as base format.

Word Processing Fields

VA FileMan word processing fields are stored similarly to multiples, and are projected by SQLI in two ways:

- As a standalone table (each line of text is one entry in the table).
- As columns for vendors who support a HUGE_CHARACTER or MEMO data type.

If you have an appropriate MEMO-like data type, you could place word-processing text into a column of this data type, and decide whether or not to make the word-processing tables available to your users.

The main problem with memo data types is that they usually come with a size constraint, and consume additional resources when you increase the maximum size. VA FileMan word processing fields, on the other hand, are unlimited in size. So you could choose a default size such as 32K for your memo-type columns. In case truncation occurs, you should return an error for word processing fields whose contents exceed your default size.

VA FileMan Indexes

VA FileMan regular-type cross references are projected by SQLI as tables. Other types of cross-references (Trigger, KWIC, MUMPS, Mnemonic, Soundex, and Bulletin) are not projected. Cross-references are primarily for vendor optimization, and should not be made available as tables to end-users.

Tables derived from cross-references use names based on the name of the indexed table followed by "_Xs_" where "s" is the index subscript, followed by the name of the column indexed (PATIENT_XB_NAME, PATIENT_XSSN_SOCIAL_SEC_NUMBER, etc.) Compression is used such that all names are no longer than 30 characters. For example:

```
PATIENT_CANCER_STATUS_CODE      (table name)
PATIENT_CANC_STAT_CODE_XB_NAME  ("B" index table name - compressed)
```

A table is projected for a cross-reference if its T_MASTER_TABLE field is populated. For multiples, there are two kinds of references, both of which are projected as tables by SQLI: regular and whole-file cross-references.

The following example shows the various parts of the table projected for a simple cross-reference for a top level file (the PATIENT file):

Table Projected for "B" Index of PATIENT File

```
NUMBER: 4650          T_NAME: PATIENT_XB_NAME
T_SCHEMA: SQLI        T_COMMENT: Index of PATIENT by NAME
T_MASTER_TABLE: PATIENT T_VERSION_FM: 1
T_UPDATE: MAY 05, 1997 T_GLOBAL: ^DPT("B", {K}, {K})
```

Table Elements Projected for PATIENT_XB_NAME

```
>D ^%G<RET>
Global ^DMSQ("E", "F", 4650
      DMSQ("E", "F", 4650
^DMSQ("E", "F", 4650, "C", 53797) =
^DMSQ("E", "F", 4650, "C", 53798) =
^DMSQ("E", "F", 4650, "P", 53796) =
```

```
NUMBER: 53796          E_NAME: PATIENT_XB_NAME_PK
E_DOMAIN: PATIENT_XB_NAME_ID E_TABLE: PATIENT_XB_NAME
E_TYPE: Primary key
E_COMMENT: Primary key header for PATIENT_XB_NAME
```

```
NUMBER: 53797          E_NAME: NAME
E_DOMAIN: CHARACTER    E_TABLE: PATIENT_XB_NAME
E_TYPE: Column
E_COMMENT: Index Primary Key #1 for PATIENT_XB_NAME.NAME
```

```
NUMBER: 53798          E_NAME: PATIENT_ID
E_DOMAIN: INTEGER      E_TABLE: PATIENT_XB_NAME
E_TYPE: Column
E_COMMENT: Index Primary Key #2 for PATIENT_XB_NAME.PATIENT_ID
```


Columns Projected for PATIENT_XB_NAME

```

>D ^%G<RET>
Global ^DMSQ("C","B",53797:53798
      DMSQ("C","B",53797:53798
^DMSQ("C","B",53797,43834) =
^DMSQ("C","B",53798,43835) =
Global ^

NUMBER: 43834          C_TABLE_ELEMENT: NAME
C_GLOBAL: ^DPT("B",

NUMBER: 43835          C_TABLE_ELEMENT: PATIENT_ID
C_PARENT: NAME         C_GLOBAL: ,

```

Primary Key Projected for PATIENT_XB_NAME

```

>D ^%G<RET>
Global ^DMSQ("P","C",53796
      DMSQ("P","C",53796
^DMSQ("P","C",53796,1,8529) =
^DMSQ("P","C",53796,2,8530) =

NUMBER: 8429          P_TBL_ELEMENT: PATIENT_XB_NAME_PK
P_COLUMN: NAME        P_SEQUENCE: 1

NUMBER: 8530          P_TBL_ELEMENT: PATIENT_XB_NAME_PK
P_COLUMN: PATIENT_ID  P_SEQUENCE: 2

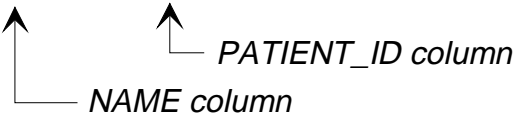
```

Partial Listing of the Index

```

^DPT("B","AMIE,TEST P",187) =
^DPT("B","BUNNY,BUGS",228) =
^DPT("B","CAMEL,HRMS",241) =

```



PATIENT_ID column

NAME column

In the example above, the primary key is a two-part key, based on two columns: the "NAME" and "PATIENT_ID" columns. The global path to "entries" in the index table is `^DPT("B", {K}, {K})`. Note that one part of the key is not ien-based, but instead is the indexed value.

For indexes whose indexed value exceeds 30 characters, a "key format" is provided that provides the transformation between the actual indexed column's field values, and the truncated-to-30 character version of the column values that appears in the index. For more information, see the description of the `SQLI_KEY_FORMAT` file.

4. File Reference

In the descriptions of SQLI files that follow, each file description contains:

- Global root of the SQLI file.
- VA FileMan data dictionary number of the SQLI file.
- All available cross references for traversing the SQLI file's entries.
- A listing of each field, with the field name, type, location, and description.
- Additional information about the purpose of the file and its fields.
- A description of the format of any code fragments supplied by this file.

Note In the tables on the following pages, SQLI field names followed by an asterisk (e.g., "S_NAME*") are never NULL when the SQLI files are populated by SQLI. This documentation convention is used to indicate that such fields are key fields for each SQLI file.

SQLI_SCHEMA File

Global Root: ^DMSQ("S",
VA FileMan Number: 1.521

Indexes:

B: ^DMSQ("S","B",\$E(S_NAME,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
S_NAME*	Free Text	0;1	Schema name (valid SQL identifier).
S_SECURITY	Free Text	1;1	Not yet implemented; for future use. M routine to check security privileges on a particular schema.
S_DESCRIPTION	Free Text	0;2	A short description of the mapped application group.

Purpose: The SQLI_SCHEMA file provides a place for SQLI to associate tables with a schema name. This allows each VA FileMan file to be automatically mapped to a schema.

Currently, SQLI automatically projects all tables as part of one schema, "SQLI". SQLI does not provide facilities for dividing VA FileMan files into separate schemas.

SQLI_KEY_WORD File

Global Root: ^DMSQ("K",
VA FileMan Number: 1.52101

Indexes:

B: ^DMSQ("K","B",\$E(KEY_WORD,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
KEY_WORD	Free Text	0;1	SQL, ODBC, or vendor keyword to reserve.

Purpose: This file is the collection point for keywords that should not be used for SQL entity names. You can add any keywords specific to your own SQL implementation through the KW^DMSQD entry point.

The SQLI_KEY_WORD file may not be populated with any key words at all. So you (the M-to-SQL vendor) should use the KW^DMSQD entry point to populate this SQLI_KEY_WORD file with:

- Any keywords specific to your (vendor) M-to-SQL product
- The standard set of reserved keywords for SQL as defined by the ANSI standard for SQL
- The keywords for ODBC as defined by Microsoft

In your instructions to sites using your SQLI mapper, make sure that adding your keywords to the SQLI_KEY_WORD file is done **prior** to the site generating their first SQLI projection.

SQLI_DATA_TYPE File

Global Root: ^DMSQ("DT",
VA FileMan Number: 1.5211

Indexes:

B: ^DMSQ("DT", "B", \$E(D_NAME, 1, 30), ien) = " "

Field Name	Type	Node; Piece	Description
D_NAME*	Free Text	0;1	Data type name (should be a valid SQL identifier).
D_COMMENT	Free Text	0;2	Brief description.
D_OUTPUT_STRATEGY	Mumps	<i>Extract Storage</i> Node 1, 1-245	Not yet implemented; for future use. Intended for future data types (pictures, formatted word processing, etc.) that VA FileMan might support in the future.
D_OUTPUT_FORMAT	Pointer to SQLI_OUTPUT_FORMAT	0;3	Not implemented in the first version of SQLI. Pointer to an Output Format to use for columns whose domains point to this data type.

Purpose: The SQLI_DATA_TYPE file is a simple list of SQL standard data types (BOOLEAN, CHARACTER, DATE, INTEGER, MEMO, MOMENT, NUMERIC, TIME) with one additional type, PRIMARY_KEY. This allows the custom VA FileMan domains in the SQLI_DOMAIN file to always be associated with a specific base SQL data type.

SQL data types determine the SQL rules for comparing values from different domains, and the operators that may be used on them. So each domain in the SQLI_DOMAIN file has an explicit SQL data type that SQL vendors should use.

Note: The PRIMARY_KEY data type (and domain) is unique to SQLI. It is used to relate primary keys to foreign keys unambiguously.

SQLI_DOMAIN File

Global Root: ^DMSQ("DM",
VA FileMan Number: 1.5212

Indexes:

B: ^DMSQ("DM", "B", \$E(DM_NAME,1,30),ien)=""
C: ^DMSQ("DM", "C", \$E(DM_TABLE,1,30),ien)=""
D: ^DMSQ("DM", "D", \$E(DM_FILEMAN_FIELD_TYPE,1,30),ien)=""
E: ^DMSQ("DM", "E", \$E(DM_DATA_TYPE,1,30),ien)=""

Field Name	Type	Node; Piece	Description
DM_NAME*	Free Text	0;1	Domain name (valid SQL identifier).
DM_DATA_TYPE*	Pointer to SQLI_DATA_TYPE	0;2	Pointer to the SQL data type to use for this domain.
DM_COMMENT	Free Text	0;3	Brief description.
DM_TABLE	Pointer to SQLI_TABLE	0;4	If this domain is for a primary or foreign key, points to the table of the primary key.
DM_WIDTH	Numeric	0;5	Maximum width of external value.
DM_SCALE	Numeric	0;6	Default number of decimal places, for NUMERIC data types only.
DM_OUTPUT_FORMAT	Pointer to SQLI_OUTPUT_FORMAT	0;7	Not implemented in the first version of SQLI. Pointer to an Output Format to use for columns that use this domain.
DM_INT_EXPR	Mumps	<i>Extract Storage</i> Node 1, 1-245	M expression to convert base value to internal (VA FileMan) format.
DM_INT_EXEC	Mumps	<i>Extract Storage</i> Node 2, 1-245	M execute statement to convert base value to internal (VA FileMan) format.
DM_BASE_EXPR	Mumps	<i>Extract Storage</i> Node 3, 1-245	M expression to convert internal (VA FileMan) value to base format.
DM_BASE_EXEC	Mumps	<i>Extract Storage</i> Node 4,	M execute statement to convert internal (VA FileMan) value to base format.

		1-245	
DM_FILEMAN_ FIELD_TYPE	Set of codes	0;8	'F' FOR FREE TEXT 'N' FOR NUMERIC 'P' FOR POINTER 'D' FOR DATE 'W' FOR WORD-PROCESSING 'K' FOR MUMPS 'C' FOR CALCULATED 'B' FOR BOOLEAN 'S' FOR SET 'V' FOR VARIABLE POINTER Original VA FileMan field type for all elements using this domain, for domains derived from VA FileMan fields. Boolean means Boolean Computed.

Purpose: Each entry in this file is a custom domain, which defines a set of values from which all objects of this domain must be drawn. In SQLI, all table elements (columns, primary keys, and foreign keys) have a domain that restricts them to their domain set.

Each domain points to a data type (from the SQLI_DATA_TYPE file) which should be used as the SQL data type for this domain. Other fields in the SQLI_DOMAIN file also constrain the set of possible values for the domain. For more information see Mapping VA FileMan Fields to SQL Data Types earlier in this chapter.

Code Fragment Formats

DM_INT_EXPR: \$S({B}="" : 0, 1 : {B})
(provide {B}, evaluates to internal FileMan form)

DM_BASE_EXPR: \$S({I} : {I}, 1 : "")
(provide {I}, evaluates to base form)

DM_INT_EXEC: S %H={B} D YMD^%DTC S {I}=X
(provide {B}, get {I} back)

DM_BASE_EXEC: N %H,X S X={I} D H^%DTC S {B}=%H
(provide {I}, get {B} back)

SQLI_KEY_FORMAT File

Global Root: ^DMSQ("KF",
VA FileMan Number: 1.5213

Indexes:

B: ^DMSQ("KF","B",\$E(KF_NAME,1,30),ien)=" "
C: ^DMSQ("KF","C",\$E(KF_DATA_TYPE,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
KF_NAME*	Free Text	0;1	Key format name.
KF_DATA_TYPE*	Pointer to SQLI_DATA_T YPE	0;2	Pointer to data type used by associated primary key (should always point to PRIMARY_KEY data type).
KF_COMMENT	Free Text	0;3	Brief description.
KF_INT_EXPR	Mumps	<i>Extract Storage</i> Node 1, 1-245	M expression to convert internal value {I} of indexed field to index primary key value {K} .
KF_INT_EXEC	Mumps	<i>Extract Storage</i> Node 2, 1-245	M executable code to set internal value {I} of indexed field to index primary key value {K} .

Purpose: Use the conversions provided in this file to translate between a column's value and the part of a primary key that uses that column. In most cases, a conversion from column value to key value is not needed.

Currently, the main situation in which a conversion is provided is for the VA FileMan indexes that are projected as tables. The index subscript is considered part of the primary key of the projected table for an index. Currently, the (regular) index subscript for a VA FileMan file is based on the field value, but is subject to truncation to 30 characters. So the value of the part of the key based on a column could differ from the value of the column itself. A standard key format is supplied and linked to all parts of primary keys that use index subscripts, whose indexed fields' maximum length exceeds 30 characters.

Code Fragment Formats

KF_INT_EXPR: \$E({I},1,30)
(provide {I}, key is returned)
KF_INT_EXEC: S {K}=\$E({I},1,30)
(provide {I}, get {K} back)

SQLI_OUTPUT_FORMAT File

Global Root: ^DMSQ("OF",
VA FileMan Number: 1.5214

Indexes:

B: ^DMSQ("OF", "B", \$E(OF_NAME, 1, 30), ien) = ""

Field Name	Type	Node; Piece	Description
OF_NAME*	Free Text	0;1	Output format name.
OF_DATA_TYPE*	Pointer to SQLI_DATA _TYPE	0;2	Pointer to the data type for which this output format applies.
OF_COMMENT	Free Text	0;3	Brief description.
OF_EXT_EXPR	Mumps	<i>Extract Storage Node 1, 1-245</i>	M expression to convert base value to external value.
OF_EXT_EXEC	Mumps	<i>Extract Storage Node 2, 1-245</i>	Will not be implemented for the first version of SQLI (patch DI*21*38). M executable code to convert base value to external value.

Purpose: Given the base column value derived from a VA FileMan field, entries in the SQLI_OUTPUT_FORMAT file provide M code to generate the external value to present to the end-user for the column in question.

Columns don't need an output format if the base column data format is the same as its external data format. Output formats are therefore provided only for columns derived from Pointer and Set of Codes VA FileMan field types.

When looking for whether an output format is provided for a column, use the column's output format if one exists. Next, check the column's domain for an output format only if one is not found for the column. Finally, check the domain's data type for an output format if one is not found for the domain.

Code Fragment Formats

OF_EXT_EXPR: `$S(' {B} : "" , 1 : $$GET^DMSQU(9.4, {B}_ " , " , .01))`
*(substitute base value for all {B} placeholders;
evaluates to external format of data).*

SQLI_TABLE File

Global Root: ^DMSQ("T",
VA FileMan Number: 1.5215

Indexes:

B: ^DMSQ("T", "B", \$E(T_NAME,1,30),ien)=" "
C: ^DMSQ("T", "C", \$E(T_FILE,1,30),ien)=" "
D: ^DMSQ("T", "D", \$E(T_GLOBAL,1,30),ien)=" "
E: ^DMSQ("T", "E", \$E(T_MASTER_TABLE,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
T_NAME*	Free Text	0;1	Table name (valid SQL identifier).
T_SCHEMA*	Pointer to SQLI_SCHE MA	0;2	Pointer to table's schema.
T_COMMENT	Free Text	0;3	Brief description.
T_MASTER_T ABLE	Pointer to SQLI_TABLE	0;4	Only populated if this table is projected for an index (it points to the indexed table.)
T_VERSION_F M	Numeric	0;5	Reserved for future use.
T_ROW_COUN T	Numeric	0;6	Estimated number of rows in the table. This field is not populated by the SQLI projection, but instead by the ALLS^DMSQS and STATS^DMSQS entry points.
T_FILE	Numeric	0;7	VA FileMan data dictionary number of file, subfile, or word processing field the table is derived from. It is null for tables that project indexes.
T_UPDATE	Date	0;8	Date table projection last updated.
T_GLOBAL	Free Text	<i>extract storage node 1, 1-245</i>	Global location of file entries. For documentation purposes only; use the C_GLOBAL values in the SQLI_COLUMN file to determine the global location of file entries in code. {K} placeholders in T_GLOBAL field values signify each part of the primary key.

Purpose: Entries in the SQLI_TABLE file project VA FileMan files, multiple fields, word processing fields, and indexes as tables.

SQLI_TABLE_ELEMENT File

Global Root: ^DMSQ("E",
VA FileMan Number: 1.5216

Indexes:

B: ^DMSQ("E","B",\$E(E_NAME,1,30),ien)=" "
C: ^DMSQ("E","C",\$E(E_DOMAIN,1,30),ien)=" "
D: ^DMSQ("E","D",\$E(E_TABLE,1,30),ien)=" "
E: ^DMSQ("E","E",\$E(E_TYPE,1,30),ien)=" "
F: ^DMSQ("E","F",E_TABLE,E_TYPE,ien)=" "
G: ^DMSQ("E","G",E_TABLE,E_NAME,ien)=" "

Field Name	Type	Node; Piece	Description
E_NAME*	Free Text	0;1	Table element name (a valid SQL identifier). Foreign keys are distinguished by the suffix _FK or _PFK, primary keys by _PK.
E_DOMAIN*	Pointer to SQLI_DOMAIN	0;2	Pointer to the domain to use for the table element.
E_TABLE*	Pointer to SQLI_TABLE	0;3	Pointer to the table the element is part of.
E_TYPE*	Set of codes	0;4	Type of table element: 'C' FOR COLUMN 'F' FOR FOREIGN KEY 'P' FOR PRIMARY KEY
E_COMMENT	Free Text	0;5	Brief description.

Purpose: In SQL Data Definition Language (DDL) a table is defined by the DDL command:

```
CREATE TABLE <table-name> (table-element-commalist)
```

There is one entry in the SQLI_TABLE_ELEMENT file for each table element (columns, primary keys, and foreign keys) that should be included in a CREATE TABLE command for each table projected in SQLI.

Entries in this file contain the two essential elements of an attribute in the relational model: attribute-name (E_NAME) and domain (E_DOMAIN). Elements not defined in the relational model, but necessary for physical mapping and formatting of table elements are contained in SQLI_COLUMN, SQLI_PRIMARY_KEY and SQLI_FOREIGN_KEY files.

SQLI_COLUMN File

Global Root: ^DMSQ("C",
VA FileMan Number: 1.5217

Indexes:

B: ^DMSQ("C","B",\$E(C_TABLE_ELEMENT,1,30),ien)=" "
C: ^DMSQ("C","C",\$E(C_PARENT,1,30),ien)=" "
D: ^DMSQ("C","D",C_FILE,C_FIELD,ien)=" "
E: ^DMSQ("C","E",\$E(C_OUTPUT_FORMAT,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
C_TBL_ELEMENT*	Pointer to SQLI_TABLE_ELEMENT	0;1	Pointer to the table element entry that this column is associated with.
C_FILE	Numeric	0;5	Corresponding VA FileMan file number, if column was derived from a data dictionary field.
C_WIDTH	Numeric	0;2	Maximum display width of column.
C_SCALE	Numeric	0;3	Default number of decimal points for NUMERIC data type only. If scale is specified as 0, the column is projected as INTEGER.
C_FIELD	Numeric	0;6	Corresponding VA FileMan field number, if column was derived from a data dictionary field.
C_NOT_NULL	Set of codes	0;7	1 if column is required in VA FileMan; 0 if not.
C_SECURE	Set of codes	0;8	Not yet implemented; for future use.
C_VIRTUAL	Set of codes	0;9	1 if column is derived from a computed field, 0 if not. If true, the corresponding field value must be retrieved using a DBS call (one is provided for this in the C_FM_EXEC field.)
C_PARENT	Pointer to SQLI_COLUMN	0;10	Populated if the global reference in the C_GLOBAL field is not a global root. Points to the column containing the next higher piece of the global reference (in C_GLOBAL) to which the current file level's key value and C_GLOBAL string should be appended to create the full global reference to the column's data. <ul style="list-style-type: none"> Null for computed field columns (no

			<p>permanent storage).</p> <ul style="list-style-type: none"> • Null for ien columns of top-level files (already at the highest level). • Null for the first index subscript column of an index table.
C_GLOBAL	Mumps	<i>Extract Storage</i> node 1, 1-245	For columns with permanent storage, partial global reference for the node where the column's data is stored.
C_PIECE	Numeric	0;11	For normally stored VA FileMan fields: The ^-delimited piece of the VA FileMan node field is stored in.
C_EXTRACT_FROM	Numeric	0;12	For extract-storage type VA FileMan fields: The first character extract position of the VA FileMan node the field is stored in.
C_EXTRACT_THRU	Numeric	0;13	For extract-storage type VA FileMan fields: The last character extract position of the VA FileMan node the field is stored in.
C_COMPUTE_EXEC	Mumps	<i>Extract Storage</i> node 2, 1-245	The internal M code VA FileMan uses to calculate a computed field's value. Warning: This code may depend on the existence of a full FileMan context; the code in C_FM_EXEC is a safer alternative.
C_FM_EXEC	Mumps	<i>Extract Storage</i> node 3, 1-245	M code to retrieve value of computed and pointer fields. Uses the DBS \$\$GET1^DIQ call to retrieve the field value.
C_POINTER	Mumps	<i>Extract Storage</i> node 4, 1-245	<p>For columns derived from set of codes fields, this field contains the pairs of internal and external forms of each code separated by semicolons. The internal and external forms of a code are separated by colons. For example:</p> <p>y: YES; n: NO;</p> <p>For columns derived from pointer fields, this field contains the global root of the referenced file. For example:</p> <p>DIC(4,</p>
C_OUTPUT_FORMAT	Pointer to SQLI_OUTPUT_FORMAT	0;4	Pointer to the output format to use for this column, if one is needed, if the external format of the data differs from the base format.

Purpose: The SQLI_COLUMN file contains the formatting and physical structure specifications for each column table element in projected tables. Each entry in the SQLI_COLUMN file has a single corresponding SQLI_TABLE_ELEMENT entry that provides the relational specifications (name and domain) for the column.

Code Fragment Formats

C_GLOBAL: *(ien columns, top-level file) ^DIZ(662000,
(ien columns, subfile) , "EX",
(VA FileMan field columns) , 0)
(Note: this field does not actually hold code, but
instead holds a global reference.)*

C_COMPUTE_EXEC: *S X=\$S(\$D(^DIA(DIA,D0,3)):^(3),1:"<deleted>")
(raw code from DD to set X to computed field value; may
require VA FileMan environment context that SQLI can't
provide - in the above example, the value of D0.)*

C_FM_EXEC: *S {V}=\$\$GET^DMSQU(9.4901,"{K3},{K2},{K1},",.03)
(uses DBS call to set the variable you substitute in
{V} to the external value of the computed or pointer
field. You must substitute appropriate iens for all {K}
placeholders to identify the entry in question.)*

SQLI_PRIMARY_KEY File

Global Root: ^DMSQ("P",
VA FileMan Number: 1.5218

Indexes:

B: ^DMSQ("P","B",\$E(P_TBL_ELEMENT,1,30),ien)=" "
C: ^DMSQ("P","C",P_TBL_ELEMENT,P_SEQUENCE,ien)=" "
D: ^DMSQ("P","D",\$E(P_COLUMN,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
P_TBL_ELEMENT*	Pointer to SQLI_TABLE_ELEMENT	0;1	Associates this part of a table's primary key with the single entry in the SQLI_TABLE_ELEMENT file that organizes the entire primary key.
P_COLUMN*	Pointer to SQLI_COLUMN	0;2	Pointer to the column on which this part of a table's primary key is based.
P_SEQUENCE*	Numeric (integer)	0;3	Sequence number of this part of the table's primary key. Use to determine what order to combine primary key columns to assemble the global path to an entry.
P_START_AT	Free Text	0;4	M literal to initialize initial subscript value for a \$ORDER loop through this part of the list of primary keys of a table.
P_END_IF	Mumps	<i>Extract Storage node 1, 1-245</i>	M expression which returns true when the \$ORDER loop started at P_START_AT reaches the end of this part of the list of primary keys of a table.
P_ROW_COUNT	Integer	0;5	Estimated number entries for this part of the primary key. For a multi-part key for the projection of a subfile, this would be set to the estimated number of entries at the file level of this part of the key. Populate this field with ALLS^DMSQS or STATS^DMSQS, after SQLI

			generation.
P_PRESELECT	Mumps	<i>Extract Storage</i> node 2, 1-245	Not implemented; for future use. Code to possibly reference files in other UCIs with extended reference syntax.
P_KEY_FORMAT	Pointer to SQLI_KEY _FORMAT	0;6	Conversion to use when the primary key value is different from the column it is based on. For primary keys of index tables, a conversion is provided to deal with the truncation of index subscripts to 30 characters.

Purpose: Each entry in the SQLI_PRIMARY_KEY file represents one part of the primary key of a projected table.

The P_COLUMN field points to the table column on which this part of the primary key is derived from.

The entire primary key of a table is composed of one or more entries in the SQLI_PRIMARY_KEY file. These entries are organized into a single key by the fact that they all point to the same single entry in the SQLI_TABLE_ELEMENT file representing the entire primary key, via the P_TBL_ELEMENT field.

Code Fragment Formats

```
P_START_AT: 0
              (value to start a $ORDER loop at, to go through a
              file's entries. Not necessarily = 0.)
P_END_IF:    '{K}
              (substitute for {K} the current ien; use to terminate a
              $ORDER loop through a file's entries. Not necessarily =
              "'{K}'".)
```


SQLI_FOREIGN_KEY File

Global Root: ^DMSQ("F",
VA FileMan Number: 1.5219

Indexes:

B: ^DMSQ("F","B",\$E(F_TBL_ELEMENT,1,30),ien)=" "

Field Name	Type	Node; Piece	Description
F_TBL_ELEMENT*	Pointer to SQLI_TABLE_ ELEMENT	0;1	Associates this part of a table's foreign key with the single entry in the SQLI_TABLE_ELEMENT file that organizes the entire foreign key.
F_PK_ELEMENT*	Pointer to SQLI_PRIMA RY_KEY	0;2	Pointer to the part of the primary key of the referenced table, that this part of the foreign key corresponds with.
F_CLM_ELEMENT *	Pointer to SQLI_COLUM N	0;3	Pointer to the column in the current table whose value should be "joined" with the associated part of the primary key of the referenced table.

Purpose: Each entry in the SQLI_FOREIGN_KEY file represents one part of a foreign key of a projected table.

As with primary keys, the entire foreign key of a table is composed of one or more entries in the SQLI_FOREIGN_KEY file. These entries are organized into a single key by pointing to the same SQLI_TABLE_ELEMENT entry, which then represents the entire foreign key.

A foreign key "pre-specifies" an explicit join between two tables. Foreign keys are projected for a table by SQLI when a join is already explicit in VA FileMan. SQLI provides foreign keys for:

- Pointer fields. For columns derived from pointer fields, a foreign key is provided for each pointer field.
- Subfiles. For table derived from subfiles, one foreign key is provided linking the subfile table to each of its "parent" tables (i.e., one to every table that represents a file level above the subfile.)

SQLI_ERROR_TEXT File

Global Root: ^DMSQ("ET",
VA FileMan Number: 1.52191

Indexes:

B: ^DMSQ("ET", "B", \$E(ERROR_TEXT, 1, 30), ien) = " "

Field Name	Type	Node; Piece	Description
ERROR_TEXT	Free Text	0;1	SQLI error message

Purpose: This file holds a list of SQLI error messages generated during the last SQLI projection. It is used by entries in the SQLI_ERROR_LOG file, to indicate which type of SQLI error occurred during SQLI generation.

Entries in this file are purged at the start of each SQLI generation. The file is then populated with only those errors that occur during the particular SQLI generation.

SQLI_ERROR_LOG File

Global Root: ^DMSQ("EX",

VA FileMan Number: 1.52192

Indexes:

B: ^DMSQ("EX", "B", \$E(FILEMAN_FILE, 1, 30), ien) = " "
 C: ^DMSQ("EX", "C", \$E(ERROR, 1, 30), ien) = " "
 D: ^DMSQ("EX", "D", \$E(ERROR_DATE, 1, 30), ien) = " "
 E: ^DMSQ("EX", "E", \$E(FILEMAN_ERROR, 1, 30), ien) = " "

Field Name	Type	Node; Piece	Description
FILEMAN_FILE	Numeric	0;1	VA FileMan file number being processed when error occurred.
FILEMAN_FIELD	Numeric	0;2	VA FileMan field number being processed when error occurred.
ERROR	Pointer to SQLI_ERROR _TEXT	0;3	Pointer to type of error.
ERROR_DATE	Date	0;4	Date of SQLI generation.
FILEMAN_ERROR	Pointer to VA FileMan DIALOG file	0;5	If the error was generated during a DBS call, and the DBS itself returned a particular error, this points to the DIALOG file reference returned by the DBS call.

Purpose: This file is a log of all errors encountered when running the SQLI generation.

You can print out the errors stored in this log directly through VA FileMan. You can also use the supplied utility, MAIN^DMSQE, to print out the errors sorted by category of error.

5. Entry Points/Supported References

SQLI provides a set of supported M routine entry points. Some entry points are intended for the use of M-to-SQL vendors; others are for general use. The supported entry points are:

Entry Point	Description
SETUP^DMSQ	Generate SQLI projection (non-interactive)
ALLF^DMSQF	Generate SQLI projection (interactive)
KW^DMSQD	Load keywords into SQLI_KEY_WORD file
ALLS^DMSQS	Generate cardinality of all tables
STATS^DMSQS	Generate cardinality of one table
\$\$CN^DMSQU	Internal SQLI naming algorithm (column)
\$\$FNB^DMSQU	Internal SQLI naming algorithm (table)
\$\$SQLI^DMSQU	Internal SQLI naming algorithm (identifier)
\$\$SQLK^DMSQU	Internal SQLI naming algorithm (identifier)

For a full description of each entry point, see the "SQLI Technical Information" chapter of the *VA FileMan SQLI Site Manual*.

In addition, all of SQLI's files, fields, and cross-references as distributed in patch DI*21*38 can be referenced directly without integration agreements. This enables M-to-SQL vendors to create SQLI mapping utilities using the SQLI file structures. Specifically, these are the files in the 1.52 to 1.53 number range, all stored in ^DMSQ.

6. Other Issues

Domain Cardinality

Most domains have no known or absolutely determinable domain cardinality. Column types for which domain cardinality can be determined are:

- Columns for Set of codes fields: Take the C_POINTER field from the column derived from the FileMan Set of codes field. `$L(C_POINTER,":")-1` yields the cardinality for this column.
- Columns for Pointer fields: Use the P_ROW_COUNT value of the primary key of the pointed-to table, or the T_ROW_COUNT of the pointed-to table. This assumes that P_ROW_COUNT and T_ROW_COUNT have been populated for the table in question using either STATS^DMSQS or ALLS^DMSQS entry points.

SQLI and Schemas

This version of SQLI projects all VA FileMan files as part of a single schema, "SQLI".

If SQLI were to project the same VA FileMan file as part of *more than one* schema, it would need to project distinct, separate entries for the file in the SQLI_TABLE file for each schema. So to project the PATIENT file in four different schemas, four different SQLI_TABLE entries would be projected, as well as four complete sets of table elements (columns, primary keys, and foreign keys).

Ordinarily it's best not to project a given file in more than one schema; in any case, SQLI currently does not support projecting the same file in multiple schemas.

SQL Identifier Naming Algorithms

By using consistent naming algorithms for files and fields, SQLI ensures that SQL table names for national files and fields between VA sites are the same. In addition, the algorithms enforce syntactical correctness and uniqueness of identifiers, and the exclusion of keywords from the naming of identifiers.

The following conventions are followed for table and table element names:

- Names are 1 to 30 characters long.
- Must start with a letter from A to z.
- May contain only the letters A through z, digits 0 through 9 and the underline character "_".
- No repeating or trailing underlines are used.
- Names are case insensitive ("a" means the same as "A").
- SQL and vendor-specific keywords may not be used as names.
- Table names must be unique within each schema.
- Table element names (column, primary key, foreign key) must be unique within each table.
- If the name is too long it is compressed by removing vowels.

Under very unusual circumstances, the naming algorithms can produce a different field or file name between sites. The known circumstances that could produce a difference are:

- The names of local files or fields result in a conflict with the naming of a national file or field.
- A difference in the excluded keyword list maintained in `SQLI_KEY_WORD` file between sites results in a naming conflict at one site, and no conflict at another.
- National packages not loaded at a particular site avoid a naming conflict that otherwise would occur.

Which Objects Are Processed Through Naming Algorithms

Tables and table element (column, primary key, and foreign key) names are generated through dynamic naming algorithms. Names for domains, data types, and output formats are manually assigned SQL-compatible names, but are not processed through the SQLI naming algorithms.

VA Business Rules and Insert/Update/Delete Operations

You may want to update VA FileMan files from SQL. Explicit support for vendors to implement Insert, Update, and Delete operations is not implemented in the first version of SQLI (patch DI*21*38).

A caution for implementing these types of access to VA FileMan data is that business rules are quite often not stored in VA FileMan data dictionaries. A significant portion of the business rules in *VISTA* applications reside in application code. Updating that doesn't go through application software cannot execute business rules stored solely in application code, and can cause data corruption by circumventing business rules.

SQLI Implementation Notes

- **.001 number fields.** The optional .001 number field for a file, if defined, represents the ien of entries. Such fields are not projected as columns by SQLI. You can access this value using the TABLE_ID column (the ien column), which SQLI does project for all tables.
- **Asterisked files.** Any files (or subfiles) whose names start with an asterisk are not projected in SQLI. Note: Adding an asterisk to the beginning of a field name is a convention in the VA Programming SAC to mark the field as obsolete.
- **Dangling pointers.** It is possible that a VA FileMan field may contain a pointer to a file not actually present at a given site. If so, the field is projected as a normal pointer field would be, but without the corresponding output format that permits navigation along a pointer chain to resolve the external value of the pointer. Such fields are flagged in the SQLI_ERROR_LOG during SQLI generation as "Pointer to Absent Files". Foreign keys for such fields are not constructed.
- **Field attributes not projected.** Along with number, the following field attributes are projected by SQLI: Label, field length, type, specifier, global subscript location, pointer, multiple-valued, and the first line of the field's description. Other field attributes, including output transforms and pointer screens, are not projected. See the *VA FileMan V. 21.0 Programmer Manual*, Global File Structure chapter, for more information about field attributes.
- **File Attributes not projected.** Only file name and number are projected. Other file attributes, such as Special Lookup and Screens, are not. See the

VA FileMan V. 21.0 Programmer Manual, Global File Structure chapter, for more information about file attributes.

- **Files not in ^DIC.** Only files with entries in ^DIC (the dictionary of files) are projected. This means only VA FileMan-compatible files are projected.
- **Internal VA FileMan tables not projected.** Certain tables used by VA FileMan internally (numbered below two) are not projected. Errors are logged during SQLI projection in the SQLI_ERROR_LOG. VA FileMan DD numbers in this category include .001, .1, .12, .15, .21, .3, 1.001 and 1.01.
- **Multiline computed fields.** Values are not returned for multiline computed fields. This is because DBS calls cannot retrieve a multiline computed field value. An example of a multiline computed field is a backward extended pointer reference.
- **Non-regular cross-references.** Only regular VA FileMan cross-references are projected. VA FileMan Trigger, KWIC (Key Word in Context), MUMPS, Mnemonic, Soundex, and Bulletin type indexes are absent from SQLI. Cross-references should not be projected to SQL end-users in any case, and only are projected for possible optimizations.
- **Output transforms.** Output transforms are not projected. If formatting needs to be applied, it can be applied at the SQL vendor column level. For more elaborate output transforms that may call routines for processing, the logic will need to be reproduced in the context of the query. Depending on your M-to-SQL product's capability, the external value of a field (after the output transform is applied) could be returned by a user-defined function that invokes the VA FileMan \$\$EXTERNAL^DILF API call.
- **Variable pointers.** Variable pointers are projected as text only. Their text value is resolved, but presented as text.

Appendix A: Quick Reference Card

Appendix A: Quick Reference Card

FILE#	FILE NAME	NODE	FIELDS (KEYS IN BOXES)	CROSS REFERENCES
1.521	SQLI_SCHEMA	^DMSQ("S",D0,0)	(#.01) S_NAME [1F]	S_NAME(B)
			(#2) S_DESCRIPTION [2F]	
		^DMSQ("S",D0,1)	(#1) S_SECURITY [1F] <i>*for future use</i>	
1.52101	SQLI_KEY_WORD	^DMSQ("K",D0,0)	(#.01) KEY_WORD [1F]	KEY_WORD(B)
1.5211	SQLI_DATA_TYPE	^DMSQ("DT",D0,0)	(#.01) D_NAME [1F]	D_NAME(B)
			(#1) D_COMMENT [2F]	
			(#3) D_OUTPUT_FORMAT [3P] <i>*for future use</i>	
		^DMSQ("DT",D0,1)	(#2) D_OUTPUT_STRATEGY [E1,245K] <i>*for future use</i>	
1.5212	SQLI_DOMAIN	^DMSQ("DM",D0,0)	(#.01) DM_NAME [1F]	DM_NAME(B)
			(#1) DM_DATA_TYPE [2P]	DM_DATA_TYPE(E)
			(#2) DM_COMMENT [3F]	
			(#3) DM_TABLE [4P]	DM_TABLE(C)
			(#4) DM_WIDTH [5N]	
			(#5) DM_SCALE [6N]	
			(#6) DM_OUTPUT_FORMAT [7P] <i>*for future use</i>	
			(#11) DM_FILEMAN_FIELD_TYPE [8S]	DM_FILEMAN_FIELD_TYPE(D)
		^DMSQ("DM",D0,1)	(#7) DM_INT_EXPR [E1,245K]	
		^DMSQ("DM",D0,2)	(#8) DM_INT_EXEC [E1,245K]	
		^DMSQ("DM",D0,3)	(#9) DM_BASE_EXPR [E1,245K]	
		^DMSQ("DM",D0,4)	(#10) DM_BASE_EXEC [E1,245K]	
1.5213	SQLI_KEY_FORMAT	^DMSQ("KF",D0,0)	(#.01) KF_NAME [1F]	KF_NAME(B)
			(#1) KF_DATA_TYPE [2P]	KF_DATA_TYPE(C)
			(#2) KF_COMMENT [3F]	
		^DMSQ("KF",D0,1)	(#3) KF_INT_EXPR [E1,245K]	
		^DMSQ("KF",D0,2)	(#4) KF_INT_EXEC [E1,245K]	
1.5214	SQLI_OUTPUT_FORMAT	^DMSQ("OF",D0,0)	(#.01) OF_NAME [1F]	OF_NAME(B)
			(#1) OF_DATA_TYPE [2P]	
			(#2) OF_COMMENT [3F]	
		^DMSQ("OF",D0,1)	(#3) OF_EXT_EXPR [E1,245K]	
		^DMSQ("OF",D0,2)	(#4) OF_EXT_EXEC [E1,245K] <i>*for future use</i>	
1.5215	SQLI_TABLE	^DMSQ("T",D0,0)	(#.01) T_NAME [1F]	T_NAME(B)
			(#1) T_SCHEMA [2P]	
			(#2) T_COMMENT [3F]	
			(#3) T_MASTER_TABLE [4P]	T_MASTER_TABLE(E)
			(#4) T_VERSION_FM [5N]	
			(#5) T_ROW_COUNT [6N]	
			(#6) T_FILE [7N]	T_FILE(C)
			(#7) T_UPDATE [8D]	
		^DMSQ("T",D0,1)	(#8) T_GLOBAL [E1,245K]	T_GLOBAL(D)

FILE#	FILE NAME	NODE	FIELDS (KEYS IN BOXES)	CROSS REFERENCES
1.5216	SQLI_TABLE_ELEMENT	^DMSQ("E",D0,0)	<div>(#.01) E_NAME [1F]</div> <div>(#1) E_DOMAIN [2P]</div> <div>(#2) E_TABLE [3P]</div> <div>(#3) E_TYPE [4S]</div> <div>(#4) E_COMMENT [5F]</div>	<div>E_NAME(B)</div> <div>E_DOMAIN(C)</div> <div>E_TABLE(D)</div> <div>E_TYPE(E)</div> <div>E_TABLE,E_NAME(G)</div> <div>E_TABLE,E_TYPE(F)</div>
1.5217	SQLI_COLUMN	^DMSQ("C",D0,0)	<div>(#.01) C_TABLE_ELEMENT [1P]</div> <div>(#2) C_WIDTH [2N]</div> <div>(#3) C_SCALE [3N]</div> <div>(#16) C_OUTPUT_FORMAT [4P]</div> <div>(#1) C_FILE [5N]</div> <div>(#4) C_FIELD [6N]</div> <div>(#5) C_NOT_NULL [7S]</div> <div>(#6) C_SECURE [8S]</div> <div>(#7) C_VIRTUAL [9S]</div> <div>(#8) C_PARENT [10P]</div> <div>(#10) C_PIECE [11N]</div> <div>(#11) C_EXTRACT_FROM [12N]</div> <div>(#12) C_EXTRACT_THRU [13N]</div> <div>(#9) C_GLOBAL [E1,245K]</div> <div>(#13) C_COMPUTE_EXEC [E1,245K]</div> <div>(#14) C_FM_EXEC [E1,245K]</div> <div>(#15) C_POINTER [E1,245K]</div>	<div>C_TABLE_ELEMENT(B)</div> <div>C_OUTPUT_FORMAT(E)</div> <div>C_FILE,C_FIELD(D)</div> <div>C_PARENT(C)</div>
1.5218	SQLI_PRIMARY_KEY	^DMSQ("C",D0,1) ^DMSQ("C",D0,2) ^DMSQ("C",D0,3) ^DMSQ("C",D0,4) ^DMSQ("P",D0,0)	<div>(#.01) P_TBL_ELEMENT [1P]</div> <div>(#1) P_COLUMN [2P]</div> <div>(#2) P_SEQUENCE [3N]</div> <div>(#3) P_START_AT [4F]</div> <div>(#5) P_ROW_COUNT [5N]</div> <div>(#7) P_KEY_FORMAT [6P]</div> <div>(#4) P_END_IF [E1,245K]</div> <div>(#6) P_PRESELECT [E1,245K] <i>*for future use</i></div>	<div>P_TBL_ELEMENT(B)</div> <div>P_COLUMN(D)</div> <div>P_TBL_ELEMENT,P_SEQUENCE(C)</div>
1.5219	SQLI_FOREIGN_KEY	^DMSQ("F",D0,0)	<div>(#.01) F_TBL_ELEMENT [1P]</div> <div>(#1) F_PK_ELEMENT [2P]</div> <div>(#2) F_CLM_ELEMENT [3P]</div>	<div>F_TBL_ELEMENT(B)</div>
1.52191	SQLI_ERROR_TEXT	^DMSQ("ET",D0,0)	<div>(#.01) ERROR_TEXT [1F]</div>	<div>ERROR_TEXT (B)</div>
1.52192	SQLI_ERROR_LOG	^DMSQ("EX",D0,0)	<div>(#.01) FILEMAN_FILE [1N]</div> <div>(#1) FILEMAN_FIELD [2N]</div> <div>(#2) ERROR [3P]</div> <div>(#3) ERROR_DATE [4D]</div> <div>(#4) FILEMAN_ERROR [5P]</div>	<div>FILEMAN_FILE(B)</div> <div>ERROR(C)</div> <div>ERROR_DATE(D)</div> <div>FILEMAN_ERROR(E)</div>

Glossary

Base Value - The stored value of a column in SQL, not transformed in any way.

Cardinality - The cardinality of a table is its number of rows; the cardinality of a domain is the number of possible values in the domain.

Column - A set of values for a particular value sequence in a row, for each row in a table (akin to a VA FileMan field). All values in a column must be of the same data type or domain.

Data Type - A set of representable values. SQL has its own set of standard data types; SQL vendors often implement additional data types.

Data Dictionary - is a file that defines a file's structure, to include a file's fields and relationships to other files.

DBA - Database Administrator for an SQL system. The DBA has, by default, full privileges to every object in the database.

DBS - Database Server. DBS is a non-interactive VA FileMan API. It makes no writes to the screen. It provides client/server access to VA FileMan data. DBS calls of particular interest to M-to-SQL vendors using SQLI include \$\$GET1^DIQ, FIELD^DID, and \$\$EXTERNAL^DILFD.

DCL - Data Control Language. The set of SQL statements through which access to the database is controlled.

DDL - Data Definition Language. The set of SQL statements through which objects are created and modified in the database.

DML - Data Manipulation Language. The set of SQL statements through which data is modified.

Domain - A set of permissible values. A domain is based on a data type, but may contain further constraints on what values are valid for the domain.

Extract Storage - When the storage location for a particular VA FileMan field is designated to be by position on a global node, instead of being character-delimited.

Field Type - The type of VA FileMan field. There are nine FileMan field types. VA FileMan field types loosely correspond to the concept of *data type*.

Foreign Key - A foreign key acts as a ready-to-use join between two tables. It matches a set of columns in one table to the primary key in another table.

Hierarchical Database - A database structure in which files can own or belong to each other. Often referred to as a parent-child structure.

Ien - Internal entry number. This is the numeric subscript beneath a file's global root under which all of the data for a given VA FileMan file entry is stored.

Ien Column - A column SQLI projects to contain the ien of a VA FileMan entry.

Join - In SQL, a join is when two or more tables are combined into a single table based on column values in an SQL SELECT statement.

M-to-SQL Product - Software that can view structured M globals as relational tables through SQL.

Multiple-Valued Field - A VA FileMan field that allows more than one value for a single entry. See also Subfile.

ODBC - Open Database Connectivity. ODBC is Microsoft's solution to enable client access to heterogeneous databases.

Outer Join - A join between two tables, where rows from one table are present in the joined table, even when there are no corresponding rows from the other table.

Output Formats - Output formats are provided by SQLI to convert column base values into a format suitable for external use by end-users.

Primary Key - a designated set of columns in a table whose values uniquely identify any row in the table.

Query - An SQL command that extracts information from an SQL database.

Relational Database - A database that is a collection of tables, and whose operations follow the relational model.

Row - A sequence of values in a table, representing one logical record.

Schema - A schema defines a portion of an SQL database as being owned by a particular user.

SQL - Structured Query Language, the predominant language and set of facilities for working with relational data. The current ANSI (American National Standards Institute) standard for SQL is X3.135-1992.

SQLI Mapper - Software written by an M-to-SQL vendor that maps the vendor's SQL data dictionaries directly to VA FileMan data, using the information projected by SQLI.

Subfile - The data structure of a multiple-valued field. In many respects, a subfile has the same characteristics as a file.

Table - A collection of rows, where each row is the equivalent of a record. A base table (one not derived from another table) is the SQL equivalent of a database file.

Table Element - a column, primary key, or foreign key that is part of a table.

View - A user-defined subset of tables, based on a SELECT statement, that contains only selected rows and columns.

.01 Field - A field that exists for every VA FileMan file, and that is used as the primary lookup value for a record.

Index

- {B} placeholder 2-1, 2-12, 3-4, 4-6, 4-8
- {E} placeholder..... 2-1
- {I} placeholder..... 2-1, 2-12, 4-6, 4-7
- {K} placeholder 2-2, 2-4, 2-8, 2-9, 2-10, 3-7, 3-8, 4-7, 4-9, 4-13, 4-15
- {V} placeholder..... 2-2, 4-13
- C_EXTRACT_FROM..... 2-11, 4-12
- C_EXTRACT_THRU..... 2-11, 4-12
- C_FM_EXEC..... 3-3, 4-12
- C_GLOBAL..... 2-10, 2-11, 4-12
- C_PIECE..... 2-11, 4-12
- C_POINTER..... 4-12
- C_VIRTUAL..... 2-11, 3-3, 4-11
- Cardinality
 - Domain..... 6-1
- Columns (processing)..... 2-6
- Computed fields..... 2-11, 3-3, 4-11
- DA Return Codes file..... 2-3
- Data dictionary synchronization 1-4
- Data storage of entries..... 2-9
- Date fields..... 3-3
- DBS callsv, 2-11, 3-3, 3-4, 4-11, 4-12, 4-18
- Delete (SQL)..... 6-3
- DM_BASE_EXEC 2-12, 3-3, 4-5, 4-6
- DM_BASE_EXPR..... 2-12, 4-5, 4-6
- DM_INT_EXEC .. 2-12, 3-3, 4-5, 4-6
- DM_INT_EXPR..... 2-12, 4-5
- Domain cardinality..... 6-1
- Domain conversions..... 2-12
- Entity - relationship diagram 1-3
- Entry data storage..... 2-9
- Entry locations..... 2-10
- Entry Points..... 5-1
- Field types (VA FileMan)..... 3-2
- Foreign keys 2-5, 2-13, 2-14, 3-4, 4-5, 4-16, 6-2
- Free text fields..... 3-3
- Identifier naming algorithms 6-2
- Ien column..... 2-7, 3-3, 4-12
- Indexes (VA FileMan)..... 3-7
- Insert (SQL)..... 6-3
- Keywords..... 4-3, 6-2
- Keywords (populating)..... 1-4
- Multiline computed fields..... 3-3
- Multiple..... *See Subfiles*
- Mumps fields..... 3-3
- Naming algorithms..... 6-2
- Numeric fields..... 3-3
- Output formats..... 2-12
- P_END_IF..... 2-9
- P_SEQUENCE..... 2-10
- P_START_AT..... 2-9
- Placeholders..... 2-1
- Pointer fields 2-11, 2-14, 3-4, 4-8, 4-16, 6-1
- Populating keywords..... 1-4
- Primary keys 2-2, 2-5, 2-7, 2-8, 2-9, 2-7-2-9, 2-10, 2-14, 3-3, 3-8, 4-5, 4-7, 4-9, 4-14, 4-15, 4-16, 6-1, 6-2
- Programming SAC..... 1-5
- Record data storage..... 2-9
- Record locations..... 2-10
- SAC (Programming)..... 1-5
- Schemas..... 2-4, 6-1
- Set of codes fields..... 3-4, 4-8, 6-1
- SQLI_COLUMN file..... 2-6, 4-11
- SQLI_DATA_TYPE file..... 4-4
- SQLI_DOMAIN file..... 4-5
- SQLI_ERROR_LOG file..... 4-18
- SQLI_ERROR_TEXT..... 4-17
- SQLI_FOREIGN_KEY file..... 4-16
- SQLI_KEY_FORMAT file..... 4-7
- SQLI_KEY_WORD file..... 1-4, 4-3
- SQLI_OUTPUT_FORMAT file .. 4-8
- SQLI_PRIMARY_KEY file 2-10, 4-14
- SQLI_SCHEMA file..... 2-4, 4-2
- SQLI_TABLE file..... 2-4, 4-9
- SQLI_TABLE_ELEMENT file 2-4, 4-10
- Subfiles 2-7, 2-8, 2-9, 2-10, 2-13, 2-14, 3-2, 3-5, 4-9, 4-13, 4-14, 4-16
- Supported References..... 5-1
- Synchronization..... 1-4
- Table elements..... 2-4

Index

Tables

Finding)	2-4
Processing)	2-4
Task Manager guidelines.....	1-5
Update (SQL).....	6-3
VA FileMan.....	v

VA FileMan field types

Summary	3-5
VA FileMan indexes	3-7
Variable pointer fields1-2, 2-11, 3-4	
Word processing fields1-2, 2-14, 3-6, 4-9	